An Isabelle Correctness Proof for the Volpano/Smith Security Typing System

Gregor Snelting and Daniel Wasserrab
IPD Snelting
Universität Karlsruhe (TH)
August 28, 2014

Abstract

The Volpano/Smith/Irvine security type systems [2] requires that variables are annotated as high (secret) or low (public), and provides typing rules which guarantee that secret values cannot leak to public output ports. This property of a program is called confidentiality.

For a simple while-language without threads, our proof shows that typeability in the Volpano/Smith system guarantees noninterference. Noninterference means that if two initial states for program execution are low-equivalent, then the final states are low-equivalent as well. This indeed implies that secret values cannot leak to public ports. For more details on noninterference and security typing systems, see [1].

The proof defines an abstract syntax and operational semantics for programs, formalizes noninterference, and then proceeds by rule induction on the operational semantics. The mathematically most intricate part is the treatment of implicit flows. Note that the Volpano/Smith system is not flow-sensitive and thus quite unprecise, resulting in false alarms. However, due to the correctness property, all potential breaks of confidentiality are discovered.
Contents

1 The Language
   1.1 Variables and Values ........................................ 3
   1.2 Expressions and Commands .................................. 3
   1.3 State ............................................................. 4
   1.4 Small Step Semantics ....................................... 4

2 Security types
   2.1 Security definitions ......................................... 10
   2.2 Lemmas concerning expressions ............................. 11
   2.3 Noninterference definitions ................................. 15
      2.3.1 Low Equivalence ....................................... 15
      2.3.2 Non Interference ...................................... 15

3 Executing the small step semantics
   3.1 An example taken from Volpano, Smith, Irvine .......... 27
theory Semantics 
imports Main 
begin 

1 The Language 

1.1 Variables and Values 

type-synonym vname = string — names for variables 

datatype val 
= Bool bool — Boolean value 
| Intg int — integer value 

abbreviation true == Bool True 
abbreviation false == Bool False 

1.2 Expressions and Commands 

datatype bop = Eq | And | Less | Add | Sub — names of binary operations 

datatype expr 
= Val val — value 
| Var vname — local variable 
| BinOp expr bop expr (-<->-[80,0,81]80) — binary operation 

Note: we assume that only type correct expressions are regarded as later 
proofs fail if expressions evaluate to None due to type errors. However there 
is [yet] no typing system 

fun binop :: bop ⇒ val ⇒ val ⇒ val option 
where 
binop Eq v1 v2 = Some(Bool(v1 = v2)) 
| binop And (Bool b1) (Bool b2) = Some(Bool(b1 ∧ b2)) 
| binop Less (Intg i1) (Intg i2) = Some(Bool(i1 < i2)) 
| binop Add (Intg i1) (Intg i2) = Some(Intg(i1 + i2)) 
| binop Sub (Intg i1) (Intg i2) = Some(Intg(i1 - i2)) 
| binop bop v1 v2 = Some(Intg(0)) 

datatype com 
= Skip 
| LAss vname expr (:=-[70,70]70) — local assignment 
| Seq com com (-;/--[61,60]60) 
| Cond expr com com (if '(' if-else [80,79,79]70) 
| While expr com (while '(' while-[80,79]70) 

fun fv :: expr ⇒ vname set — free variables in an expression 
where
\[ FVc: \text{fv} (\text{Val} V) = \{\} \]
\[ | \text{FVu: \text{fv} (\text{Var} V) = \{V\} } \]
\[ | \text{FVe: \text{fv} (e_1 \text{ \textless} \text{bop} \text{\textgreater} e_2) = \text{fv} e_1 \cup \text{fv} e_2} \]

1.3 State

type-synonym \text{state} = \text{vname} \rightarrow \text{val}

\text{interpret} silently assumes type correct expressions, i.e. no expression evaluates to None

\textbf{fun} \text{interpret} :: \text{expr} \Rightarrow \text{state} \Rightarrow \text{val option} ([\ldots])
\textbf{where}
\text{Val}: [\text{Val} v] s = \text{Some} v
\text{Var}: [\text{Var} V] s = s V
\text{BinOp}: [\text{e}_1 \text{\textless} \text{bop} \text{\textgreater} \text{e}_2] s = (\text{case [\text{e}_1] s of None ⇒ None} | \text{Some} v_1 ⇒ (\text{case [\text{e}_2] s of None ⇒ None} | \text{Some} v_2 ⇒ \text{binop bop} v_1 v_2))

1.4 Small Step Semantics

\textbf{inductive} \text{red} :: \text{com} * \text{state} ⇒ \text{com} * \text{state} ⇒ \text{bool}
\textbf{and} \text{red’} :: \text{com} ⇒ \text{state} ⇒ \text{com} ⇒ \text{state} ⇒ \text{bool}
((((1\langle-,/-\rangle) ⇒/ (1\langle-,/-\rangle)) [0,0,0,0] \text{SI}) \text{ where}
\langle c_1,s_1 \rangle \rightarrow \langle c_2,s_2 \rangle = = \text{red} (c_1,s_1) (c_2,s_2) |
\text{RedLAss:}
\langle V:=e,s \rangle \rightarrow \langle \text{Skip},s(V:=[e] s) \rangle)
| \text{SeqRed:}
\langle c_1,s \rangle \rightarrow \langle c_1',s' \rangle \Rightarrow \langle c_1'';c_2,s' \rangle \rightarrow \langle c_1',c_2,s' \rangle
| \text{RedSeg:}
\langle \text{Skip};;c_2,s \rangle \rightarrow \langle c_2,s \rangle
| \text{RedCondTrue:}
\langle b \rangle s = \text{Some} true \Rightarrow \langle \text{if} (b) \ c_1 \text{ else} \ c_2,s \rangle \rightarrow \langle c_1,s \rangle
| \text{RedCondFalse:}
\langle b \rangle s = \text{Some} false \Rightarrow \langle \text{if} (b) \ c_1 \text{ else} \ c_2,s \rangle \rightarrow \langle c_2,s \rangle
| \text{RedWhileTrue:}
\langle b \rangle s = \text{Some} true \Rightarrow \langle \text{while} (b) \ c,s \rangle \rightarrow \langle c;\text{while} (b) \ c,s \rangle
| \text{RedWhileFalse:}
\langle b \rangle s = \text{Some} false \Rightarrow \langle \text{while} (b) \ c,s \rangle \rightarrow \langle \text{Skip},s \rangle

\textbf{lemmas} \text{red-induct} = \text{red.induct\{split-format (complete)\]}
\textbf{abbreviation} \text{reds ::com ⇒ state ⇒ com ⇒ state ⇒ bool}
((((1\langle-,/-\rangle) ⇒/ (1\langle-,/-\rangle)) [0,0,0,0] \text{SI}) \text{ where}
\langle c,s \rangle \rightarrow* \langle c',s' \rangle = = \text{red}^* (c,s) (c',s')
lemma *Skip-reds:*

\[(\text{Skip}, s) \rightarrow^* (c', s') \implies s = s' \land c' = \text{Skip}\]

by (blast elim: converse-tranclpE red.cases)

lemma *LAss-reds:*

\[(V := e, s) \rightarrow^* (\text{Skip}, s') \implies s' = s(V := [e] s)\]

proof (induct \(V := e, s\) rule: converse-tranclp-induct2)

\[\text{case (step } s \ c'' \ s''')\]

hence \(c'' = \text{Skip}\) and \(s''' = s(V := ([e] s))\) by (auto elim: red.cases)

with \(\langle c'', s''' \rangle \rightarrow^* (\text{Skip}, s')\)

show ?case by (auto dest: Skip-reds)

qed

lemma *Seq2-reds:*

\[(\text{Skip}; c_2, s) \rightarrow^* (\text{Skip}, s') \implies (c_2, s) \rightarrow^* (\text{Skip}, s')\]

by (induct \(c = \text{Skip}; c_2, s\) rule: converse-tranclp-induct2) (auto elim: red.cases)

lemma *Seq-reds:*

assumes \((c_1; c_2, s) \rightarrow^* (\text{Skip}, s')\)

obtains \(s''\) where \((c_1, s) \rightarrow^* (\text{Skip}, s'')\) and \((c_2, s') \rightarrow^* (\text{Skip}, s'')\)

proof -

have \(\exists \ s''\). \((c_1, s) \rightarrow^* (\text{Skip}, s'') \land (c_2, s') \rightarrow^* (\text{Skip}, s'')\)

proof -

\{ fix \(c\ c'\)

assume \((c, s) \rightarrow^* (c', s')\) and \(c = c_1; c_2\) and \(c' = \text{Skip}\)

hence \(\exists \ s''\). \((c_1, s) \rightarrow^* (\text{Skip}, s'') \land (c_2, s') \rightarrow^* (\text{Skip}, s'')\)

proof (induct arbitrary: \(c_1\) rule: converse-tranclp-induct2)

\[\text{case refl thus ?case by simp}\]

next

\[\text{case (step } s \ c'' \ s''')\]

note \(IH = \langle \forall c_1. [c'' = c_1; c_2; c' = \text{Skip}]\)

\[\implies \exists \ sx. \langle c_1, s''\rangle \rightarrow^* (\text{Skip}, sx) \land (c_2, sx) \rightarrow^* (\text{Skip}, s'')\]

from step

have \((c_1; c_2, s) \rightarrow (c'', s''')\) by simp

hence \((c_1 = \text{Skip} \land c'' = c_2 \land s = s'')\) \lor

\((\exists c_1'. \langle c_1', s''\rangle \rightarrow (c_1', s''') \land c''' = c_1'; c_2)\)

by (auto elim: red.cases)

thus ?case

proof

assume \((c_1 = \text{Skip} \land c'' = c_2 \land s = s'')\)

with \((c'', s''') \rightarrow^* (c', s')\)

\(c' = \text{Skip}\)

show ?thesis by auto

next

assume \(\exists c_1'. \langle c_1, s \rangle \rightarrow (c_1', s''') \land c''' = c_1'; c_2\)

then obtain \(c_1'\) where \((c_1, s) \rightarrow (c_1', s''')\) and \(c''' = c_1'; c_2\) by blast

from \(IH[OF \ (c'' = c_1'; c_2) \ c' = \text{Skip}]\)

obtain \(sx\) where \((c_1', s'') \rightarrow^* (\text{Skip}, sx)\) and \((c_2, sx) \rightarrow^* (\text{Skip}, s'')\)

by blast

from \((c_1, s) \rightarrow (c_1', s''')\)

\(\langle c_1', s''''\rangle \rightarrow^* (\text{Skip}, sx)\)

qed
have \((c_1, s) \rightarrow^* \langle \text{Skip}, s' \rangle\) by (auto intro:converse-rtranclp-into-rtranclp)
with \(\langle c_2, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle\) show ?thesis by auto
qed

qed 

with \(\langle c_1; c_2, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle\) show ?thesis by simp
qed

with that show ?thesis by blast
qed

lemma Cond-True-or-False:
\(\langle \text{if} \ (b) \ c_1 \ \text{else} \ c_2, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle \Longrightarrow \ [b] \ s = \text{Some true} \lor \ [b] \ s = \text{Some false}\)
by (induct c::=\(\text{if} \ (b) \ c_1 \ \text{else} \ c_2 \ s \ \text{rule:converse-rtranclp-induct2}\))(auto elim:red.cases)

lemma CondTrue-reds:
\(\langle \text{if} \ (b) \ c_1 \ \text{else} \ c_2, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle \Longrightarrow \ [b] \ s = \text{Some true} \Longrightarrow \ (c_1, s) \rightarrow^* \langle \text{Skip}, s' \rangle\)
by (induct c::=\(\text{if} \ (b) \ c_1 \ \text{else} \ c_2 \ s \ \text{rule:converse-rtranclp-induct2}\))(auto elim:red.cases)

lemma CondFalse-reds:
\(\langle \text{if} \ (b) \ c_1 \ \text{else} \ c_2, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle \Longrightarrow \ [b] \ s = \text{Some false} \Longrightarrow \ (c_2, s) \rightarrow^* \langle \text{Skip}, s' \rangle\)
by (induct c::=\(\text{if} \ (b) \ c_1 \ \text{else} \ c_2 \ s \ \text{rule:converse-rtranclp-induct2}\))(auto elim:red.cases)

lemma WhileFalse-reds:
\(\langle \text{while} \ (b) \ cx, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle \Longrightarrow \ [b] \ s = \text{Some false} \Longrightarrow \ s = s'\)

proof (induct while \(b\) \ cx \ s \ \text{rule:converse-rtranclp-induct2})
case step thus ?case by (auto elim:red.cases dest: Skip-reds)
qed

lemma WhileTrue-reds:
\(\langle \text{while} \ (b) \ cx, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle \Longrightarrow \ [b] \ s = \text{Some true} \Longrightarrow \exists \sz. \ (cx, s) \rightarrow^* \langle \text{Skip}, s' \rangle \ \land \ \langle \text{while} \ (b) \ cx, \sz \rangle \rightarrow^* \langle \text{Skip}, s' \rangle\)

proof (induct while \(b\) \ cx \ s \ \text{rule:converse-rtranclp-induct2})
case (step s c'' s'')
hence c''' = cx;\while (b) \ cx \ \land \ s''' = s\ by (auto elim:red.cases)
with \(\langle c''', s''' \rangle \rightarrow^* \langle \text{Skip}, s' \rangle\) show ?case by (auto dest:Seq-reds)
qed

lemma While-True-or-False:
\(\langle \text{while} \ (b) \ \text{com}, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle \Longrightarrow \ [b] \ s = \text{Some true} \lor \ [b] \ s = \text{Some false}\)
by (induct c::=\(\text{while} \ (b) \ \text{com} \ s \ \text{rule:converse-rtranclp-induct2}\))(auto elim:red.cases)

inductive red-n :: \text{com} \Rightarrow \text{state} \Rightarrow \text{nat} \Rightarrow \text{com} \Rightarrow \text{state} \Rightarrow \text{bool} 
\((\forall (\langle c, s \rangle) \rightarrow^* (\langle c', s' \rangle)) \ [0, 0,0, 0] \Rightarrow\) 81
where red-n-Base: \(\langle c, s \rangle \rightarrow^0 \langle c, s \rangle\)

| red-n-Rec: \(\langle c, s \rangle \rightarrow \langle c'', s'' \rangle; \langle c', s' \rangle \rightarrow^* \langle c', s' \rangle\) \Longrightarrow \langle c, s \rangle \rightarrow^* \text{Suc} \ n \langle c', s' \rangle

lemma Seq-red-nE: assumes \(\langle c_1; c_2, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle\)
obtains $ij s''$ where $\langle c_1, s \rangle \rightarrow^i \langle \text{Skip}, s' \rangle$ and $\langle c_2, s'' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle$
and $n = i + j + 1$

proof –
from $\langle c_1 :: c_2, s \rangle \rightarrow^n \langle \text{Skip}, s' \rangle$

have $\exists i j s''. \langle c_1, s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \land \langle c_2, s'' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \land n = i + j + 1$

proof (induct $c_1 :: c_2$ s n $\text{Skip}$ s' arbitrary; $c_1$ rule: red-n.induct)
case (red-n-Rec $s$ $c''$ $s''$ $n$ $s'$)

note $IH = \{ c_1, c'' = c_1 :: c_2 \}

\implies \exists i j sx. \langle c_1, s'' \rangle \rightarrow^i \langle \text{Skip}, sx \rangle \land \langle c_2, sx \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \land n = i + j + 1$
from $\langle c_1 :: c_2, s \rangle \rightarrow \langle c''', s''' \rangle$

have $(c_1 = \text{Skip} \land c'' = c_2 \land s = s'') \lor (\exists c_1', c'' = c_1 :: c_2 \land \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle)$
by (induct $c_1 :: c_2$ - - rule: red-induct) auto
thus ?case

proof
assume $c_1 = \text{Skip} \land c'' = c_2 \land s = s''$
hence $c_1 = \text{Skip}$ and $c'' = c_2 \land s = s''$ by simp-all
from $\langle c_1 = \text{Skip} \rangle$ have $\langle c_1, s \rangle \rightarrow \langle \text{Skip}, s' \rangle$ by (fastforce intro: red-n.Base)
with $\langle c''', s''' \rangle \rightarrow^n \langle \text{Skip}, s' \rangle$, $c'' = c_2$, $s = s''$

show ?thesis by (rule_tac x = 0 in exI) auto

next
assume $\exists c_1'. c'' = c_1' :: c_2 \land \langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle$
then obtain $c_1'$ where $c'' = c_1 :: c_2$ and $\langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle$ by blast
from IH[OF $c'' = c_1 :: c_2$] obtain $i j sx$

where $\langle c_1', s'' \rangle \rightarrow^i \langle \text{Skip}, sx \rangle$ and $\langle c_2, sx \rangle \rightarrow^j \langle \text{Skip}, s' \rangle$

and $n = i + j + 1$ by blast
from $\langle c_1, s \rangle \rightarrow \langle c_1', s'' \rangle$, $\langle c_1', s'' \rangle \rightarrow^i \langle \text{Skip}, sx \rangle$

have $\langle c_1, s \rangle \rightarrow \langle \text{Suc i, Skip}, sx \rangle$ by (rule red-n.red-n-Rec)
with $\langle c_2, sx \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \land n = i + j + 1$ show ?thesis
by (rule_tac x = Suc i in exI) auto

qed

with that show ?thesis by blast

qed

lemma while-red-nE:
$\langle \text{while (b) cx}, s \rangle \rightarrow^n \langle \text{Skip}, s' \rangle$
$\implies (\exists i j s''. \langle b, s = \text{Some False} \land s = s' \land n = 1 \rangle \lor
(\exists i j s'. \langle b, s = \text{Some True} \land \langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \land
\langle \text{while (b) cx, s''} \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \land n = i + j + 2)$

proof (induct while (b) cx s n \text{Skip} s' rule: red-n.induct)
case (red-n-Rec $s$ $c''$ $s''$ $n$ $s'$)
from $\langle \text{while (b) cx, s} \rangle \rightarrow \langle c'', s'' \rangle$

have $\langle b, s = \text{Some False} \land c'' = \text{Skip} \land s'' = s \rangle \lor
(\exists b, s = \text{Some True} \land c'' = \text{cx}; \langle \text{while (b) cx} \land s'' = s \rangle$
by (induct while (b) cx - - rule: red-induct) auto
thus ?case

proof
assume \([b]\) \(s = \text{Some false} \land c'' = \text{Skip} \land s'' = s\)

hence \([b]\) \(s = \text{Some false} \land c'' = \text{Skip} \land s'' = s\) by simp-all

with \(\langle c'', s'' \rangle \rightarrow^n \langle \text{Skip}, s' \rangle\) have \(s = s'\) and \(n = 0\)
by (induct - \(-\text{Skip} - \text{rule:red-n-induct,auto elim:red.cases}\)

with \([b]\) \(s = \text{Some false}\) show \(?\text{thesis}\) by fastforce

next

assume \([b]\) \(s = \text{Some true} \land c'' = cx\); while \((b)\) cx \land s'' = s

hence \([b]\) \(s = \text{Some true} \land c'' = cx\); while \((b)\) cx

and \(s'' = s\) by simp-all

with \(\langle c'', s'' \rangle \rightarrow^n \langle \text{Skip}, s' \rangle\)

obtain \(i j s x\) where \(\langle cx, s \rangle \rightarrow^i \langle \text{Skip}, sx \rangle\) and \((\text{while } (b) cx, sx) \rightarrow^j \langle \text{Skip}, s' \rangle\)

and \(n = i + j + 1\) by (fastforce elim: Seq-red-nE)

with \([b]\) \(s = \text{Some true}\) show \(?\text{thesis}\) by fastforce

qed

qed

lemma \text{while-red-n-induct} \{ consumes 1, case-names false true \}:

assumes major: \(\langle \text{while } (b) cx, s \rangle \rightarrow^n \langle \text{Skip}, s' \rangle\)

and \(\text{IHfalse}\): \(\forall s. \ [b] s = \text{Some false} \Rightarrow P s s\)

and \(\text{IHtrue}\): \(\forall i j s''. \ [b] s = \text{Some true}; \langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle; (\text{while } (b) cx, s) \rightarrow^j \langle \text{Skip}, s' \rangle; P s s \equiv P s s'\)

shows \(P s s'\)

using major

proof (induct \(n\) arbitrary; \(s\) rule: nat-less-induct)

fix \(n\) \(s\)

assume \(\text{IHall}\): \(\forall m < n. \forall x. \langle \text{while } (b) cx, x \rangle \rightarrow^m \langle \text{Skip}, s' \rangle \Rightarrow P x s'\)

and \(\langle \text{while } (b) cx, s \rangle \rightarrow^n \langle \text{Skip}, s' \rangle\)

from \(\langle \text{while } (b) cx, s \rangle \rightarrow^n \langle \text{Skip}, s' \rangle\)

have \([b] s = \text{Some false} \land s = s' \land n = 1\) \lor
\((\exists i j s''. \ [b] s = \text{Some true}; \langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \land (\text{while } (b) cx, s) \rightarrow^j \langle \text{Skip}, s' \rangle \land n = i + j + 2)\)

by (rule while-red-nE)

thus \(P s s'\)

proof

assume \([b]\) \(s = \text{Some false} \land s = s' \land n = 1\)

hence \([b]\) \(s = \text{Some false} \land s = s'\) by auto

from \(\text{IHfalse}\) \(\text{OF}\) \([b]\) \(s = \text{Some false}\) have \(P s s\).

with \(s = s'\) show \(?\text{thesis}\) by simp

next

assume \(\exists i j s''. \ [b] s = \text{Some true} \land \langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle \land (\text{while } (b) cx, s) \rightarrow^j \langle \text{Skip}, s' \rangle \land n = i + j + 2\)

then obtain \(i j s''\) where \([b] s = \text{Some true}\)

and \(\langle cx, s \rangle \rightarrow^i \langle \text{Skip}, s'' \rangle\) and \((\text{while } (b) cx, s) \rightarrow^j \langle \text{Skip}, s' \rangle\)

and \(n = i + j + 2\) by blast

with \(\text{IHall}\) have \(P s s'' s'\)

apply (erule_tac \(x = j\) in allE) apply clarsimp done

from \(\text{IHtrue}\) \(\text{OF}\) \([b]\) \(s = \text{Some true}\) \(\langle cx, s \rangle \rightarrow^j \langle \text{Skip}, s'' \rangle\)
\langle \text{while} \ (b) \ cx, s' \rangle \rightarrow^j \langle \text{Skip}, s' \rangle \text{ this } \text{show} \ ?\text{thesis}.

\text{qed}
\text{qed}

\text{lemma} \ \text{reds-to-red-n} : (c, s) \rightarrow^* \langle c', s' \rangle \implies \exists n \ (c, s) \rightarrow^n \langle c', s' \rangle
\text{by} (\text{induct rule}; \text{converse-rtranclp-induct2}; \text{auto intro}; \text{red-n intros})

\text{lemma} \ \text{red-n-to-reds} : (c, s) \rightarrow^n \langle c', s' \rangle \implies (c, s) \rightarrow \langle c', s' \rangle
\text{by} (\text{induct rule}; \text{red-n.induct}; \text{auto intro}; \text{converse-rtranclp-into-rtranclp})

\text{lemma} \ \text{while-reds-induct} \{\text{consumes 1}, \text{case-names false true}\}:
\langle \text{while} \ (b) \ cx, s \rangle \rightarrow^* \langle \text{Skip}, s' \rangle; \ \forall s \ [b] s = \text{Some false} \implies P \ s \ s;
\langle \text{while} \ (b) \ cx, s'' \rangle \rightarrow^* \langle \text{Skip}, s'' \rangle;
\langle \text{while} \ (b) \ cx, s', s'' \rangle \rightarrow^* \langle \text{Skip}, s'' \rangle; \ P \ s'' \ s'] \implies P \ s \ s'
\text{apply} (\text{drule} \ \text{reds-to-red-n} \ \text{clarsimp})
\text{apply} (\text{erule} \ \text{while-red-n-induct} \ \text{clarsimp})
\text{by} (\text{auto dest}; \text{red-n-to-reds})

\text{lemma} \ \text{red-det}:
\langle c, s \rangle \rightarrow \langle c_1, s_1 \rangle; \ \langle c, s \rangle \rightarrow \langle c_2, s_2 \rangle \implies c_1 = c_2 \land s_1 = s_2
\text{proof} (\text{induct arbitrary}; c_2 \ \text{rule}; \text{red-induct})
\text{case} \ (\text{SeqRed} \ c_1 \ s \ c_1' \ s_1' \ c_2')
\text{note} \ IH = (\langle c_1, s \rangle \rightarrow \langle c_2, s_2 \rangle \implies c_1' = c_2 \land s_1' = s_2)
\text{from} \ \langle c_1; c_2', s \rangle \rightarrow \langle c_2, s_2 \rangle \ \text{have} \ c_1 = \text{Skip} \lor (\exists cx. \ c_2 = cx; c_2' \land c_1 \rightarrow \langle cx, s_2 \rangle)
\text{by} (\text{fastforce elim}; \text{red.cases})
\text{thus} \ ?\text{case}
\text{proof}
\text{assume} \ c_1 = \text{Skip}
\text{with} \ \langle c_1, s \rangle \rightarrow \langle c_1', s_1' \rangle \ \text{have} \ \text{False} \ \text{by} (\text{fastforce elim}; \text{red.cases})
\text{thus} \ ?\text{thesis} \ \text{by} \ \text{simp}
\text{next}
\text{assume} \ \exists cx. \ c_2 = cx; c_2' \land (c_1, s) \rightarrow \langle cx, s_2 \rangle
\text{then obtain} \ cx \ \text{where} \ c_2 = cx; c_2' \ \text{and} \ \langle c_1, s \rangle \rightarrow \langle cx, s_2 \rangle \ \text{by} \ \text{blast}
\text{from} \ \text{IH} (\langle c_1, s \rangle \rightarrow \langle cx, s_2 \rangle) \ \text{have} \ c_1' = cx \land s_2 \ \text{.}
\text{with} \ c_2 = cx; c_2' \ \text{show} \ ?\text{thesis} \ \text{by} \ \text{simp}
\text{qed}
\text{qed} (\text{fastforce elim}; \text{red.cases} +)

\text{theorem} \ \text{reds-det}:
\langle c, s \rangle \rightarrow^* \langle \text{Skip}, s_1 \rangle; \ \langle c, s \rangle \rightarrow^* \langle \text{Skip}, s_2 \rangle \implies s_1 = s_2
\text{proof} (\text{induct rule}; \text{converse-rtranclp-induct2})
\text{case refl}
from $(\text{Skip}, s_1) \rightarrow^* (\text{Skip}, s_2)$ show ?case
by $(\text{erule converse-rtranclpE, auto elim:red.cases})$

next

case $(\text{step} \ c''', s''', c', s')$

note $IH = (\langle c', s' \rangle \rightarrow^* (\text{Skip}, s_2) \implies s_1 = s_2)$
from step have $(c''', s''') \rightarrow (c', s')$
by simp
from $(c''', s''') \rightarrow^* (\text{Skip}, s_2)$ this have $(c', s') \rightarrow^* (\text{Skip}, s_2)$
by $(\text{erule converse-rtranclpE, auto elim:red.cases dest:red-det})$
from $IH[\text{OF this}]$ show ?thesis .

qed

end

theory secTypes
imports Semantics
begin

2 Security types

2.1 Security definitions

datatype secLevel = Low | High

type-synonym typeEnv = vname $\rightarrow$ secLevel

inductive secExprTyping :: typeEnv $\Rightarrow$ expr $\Rightarrow$ secLevel $\Rightarrow$ bool (- $\vdash$ - $\vdash$ -)
where typeVal: $\Gamma \vdash \text{Val} \ V : \text{lev}$
| typeVar: $\Gamma \vdash \text{Var} \ V n : \text{lev}$
| typeBinOp1: $[\Gamma \vdash e_1 : \text{Low}; \Gamma \vdash e_2 : \text{Low}] \implies \Gamma \vdash e_1 \leftarrow\ bop\rightarrow e_2 : \text{Low}$
| typeBinOp2: $[\Gamma \vdash e_1 : \text{High}; \Gamma \vdash e_2 : \text{lev}] \implies \Gamma \vdash e_1 \leftarrow\ bop\rightarrow e_2 : \text{High}$
| typeBinOp3: $[\Gamma \vdash e_1 : \text{lev}; \Gamma \vdash e_2 : \text{High}] \implies \Gamma \vdash e_1 \leftarrow\ bop\rightarrow e_2 : \text{High}$

inductive secComTyping :: typeEnv $\Rightarrow$ secLevel $\Rightarrow$ com $\Rightarrow$ bool (,- $\vdash$ - $\vdash$ -)
where typeSkip: $\Gamma, T \vdash \text{Skip}$
| typeAssH: $\Gamma \ V n = \text{Some \ High} \implies \Gamma, T \vdash V := e$
| typeAssL: $[\Gamma \vdash e : \text{Low}; \Gamma \ V n = \text{Some \ Low}] \implies \Gamma, \text{Low} \vdash V := e$
| typeSeq: $[\Gamma, T \vdash c_1; \Gamma, T \vdash c_2] \implies \Gamma, T \vdash c_1 \quad c_2$
| typeWhile: $[\Gamma \vdash b : T; \Gamma, T \vdash c] \implies \Gamma, T \vdash \text{while} \ (b) \ c$
| typeIf: | \[ \Gamma \vdash b : T; \Gamma, T \vdash c_1; \Gamma, T \vdash c_2 \] \implies \Gamma, T \vdash \text{if} (b) c_1 \text{ else } c_2 |
| typeConvert: | \Gamma, \text{High} \vdash c \implies \Gamma, \text{Low} \vdash c |

2.2 Lemmas concerning expressions

lemma exprTypeable:
assumes \(fv e \subseteq \text{dom } \Gamma\) obtains \(T\) where \(\Gamma \vdash e : T\)
proof -
from \(fv e \subseteq \text{dom } \Gamma\) have \(\exists T. \Gamma \vdash e : T\)
proof(pinduct \(e\))
case \(\text{Val } \mathit{V}\)
have \(\Gamma \vdash \text{Val } \mathit{V} : \text{Low}\) by (rule typeVal)
thus \(?\text{case}\) by (rule exI)
next
case \(\text{Var } \mathit{V}\)
have \(\mathit{V} \in \text{fv } (\text{Var } \mathit{V})\) by simp
with \(\text{fv } (\text{Var } \mathit{V}) \subseteq \text{dom } \Gamma\) have \(\mathit{V} \in \text{dom } \Gamma\) by simp
then obtain \(T\) where \(\Gamma \vdash \mathit{V} = \text{!} T\) by auto
hence \(\Gamma \vdash \text{Var } \mathit{V} : T\) by (rule typeVar)
thus \(?\text{case}\) by (rule exI)
next
case \(\text{BinOp } e_1 \mathit{bop} e_2\)
note \(IH1 = (fv e_1 \subseteq \text{dom } \Gamma \implies \exists T. \Gamma \vdash e_1 : T)\)
note \(IH2 = (fv e_2 \subseteq \text{dom } \Gamma \implies \exists T. \Gamma \vdash e_2 : T)\)
from \(fv (e_1 \mathit{bop} e_2) \subseteq \text{dom } \Gamma\) have \(\text{fv } e_1 \subseteq \text{dom } \Gamma\) and \(fv e_2 \subseteq \text{dom } \Gamma\) by auto
from \(IH1[\text{OF } (fv e_1 \subseteq \text{dom } \Gamma)]\) obtain \(T1\) where \(\Gamma \vdash e_1 : T_1\) by auto
from \(IH2[\text{OF } (fv e_2 \subseteq \text{dom } \Gamma)]\) obtain \(T2\) where \(\Gamma \vdash e_2 : T_2\) by auto
show \(?\text{case}\)
proof (cases \(T1\))
case \(\text{Low}\)
show \(?\text{thesis}\)
proof (cases \(T2\))
case \(\text{Low}\)
with \(\Gamma \vdash e_1 : T_1\) \(\langle \Gamma \vdash e_2 : T_2\rangle (T_1 = \text{Low})\)
have \(\Gamma \vdash e_1 \mathit{bop} e_2 : \text{Low}\) by (simp add: typeBinOp1)
thus \(?\text{thesis}\) by (rule exI)
next
case \(\text{High}\)
with \(\Gamma \vdash e_1 : T_1\) \(\langle \Gamma \vdash e_2 : T_2\rangle (T_1 = \text{Low})\)
have \(\Gamma \vdash e_1 \mathit{bop} e_2 : \text{High}\) by (simp add: typeBinOp3)
thus \(?\text{thesis}\) by (rule exI)
qed
next
case \(\text{High}\)
with \(\Gamma \vdash e_1 : T_1\) \(\langle \Gamma \vdash e_2 : T_2\rangle\)
have \(\Gamma \vdash e_1 \mathit{bop} e_2 : \text{High}\) by (simp add: typeBinOp2)
thus \(?thesis\) by \((rule\ exI)\)

qed

qed

with \(?thesis\) by \textit{blast}

qed


\textbf{lemma \textit{exprBinopTypeable}}:
\assumes \(\Gamma \vdash e_1 \langle\textit{bop}\rangle e_2 : T\)
\shows \((\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2)\)
\using \textit{assms} by \(\text{(auto elim:secExprTyping.cases)}\)

\textbf{lemma \textit{exprTypingHigh}}:
\assumes \(\Gamma \vdash e : T\) and \(x \in \text{fv} e\) and \(\Gamma \vdash x = \text{Some High}\)
\shows \(\Gamma \vdash e : \text{High}\)
\using \textit{assms}
\proof \text{(induct \(e\) arbitrary; \(T\))}
\case \((\text{Val} V)\) \show \(?case\) by \(\text{(rule typeVal)}\)
\next \case \((\text{Var} V)\)
\from \((x \in \text{fv} (\text{Var} V))\) \have \(x = V\) by \textit{simp}
\with \(\Gamma \vdash x = \text{Some High}\) \show \(?case\) (\textit{add: typeVar})
\next \case \((\text{BinOp} e_1 \text{ bop} e_2)\)
\note \(IH1 = \exists T. (\Gamma \vdash e_1 : T; x \in \text{fv} e_1; \Gamma \vdash x = \text{Some High}) \Rightarrow \Gamma \vdash e_1 : \text{High}\)
\note \(IH2 = \exists T. (\Gamma \vdash e_2 : T; x \in \text{fv} e_2; \Gamma \vdash x = \text{Some High}) \Rightarrow \Gamma \vdash e_2 : \text{High}\)
\from \((\Gamma \vdash e_1 \langle\textit{bop}\rangle e_2 : T)\)
\have \(T; (\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2)\) by \(\text{(auto intro!:\textit{exprBinopTypeable})}\)
\then \obtain \(T_1\) where \(\Gamma \vdash e_1 : T_1\) by \textit{auto}
\from \(T\) \obtain \(T_2\) where \(\Gamma \vdash e_2 : T_2\) by \textit{auto}
\from \((x \in \text{fv} (e_1 \langle\textit{bop}\rangle e_2))\) \have \(x \in (\text{fv} e_1 \cup \text{fv} e_2)\) by \textit{simp}
\hence \(x \in \text{fv} e_1 \lor x \in \text{fv} e_2\) by \textit{auto}
\thus \(?case\)
\proof
\assume \(x \in \text{fv} e_1\)
\from \(IH1\) \((\Gamma \vdash e_1 : T_1)\) this \(\Gamma \vdash x = \text{Some High}\) \have \(\Gamma \vdash e_1 : \text{High}\).
\with \(\Gamma \vdash e_2 : T_2\) \show \(?thesis\) by \(\text{(simp add: typeBinOp2)}\)
\next \assume \(x \in \text{fv} e_2\)
\from \(IH2\) \((\Gamma \vdash e_2 : T_2)\) this \(\Gamma \vdash x = \text{Some High}\) \have \(\Gamma \vdash e_2 : \text{High}\).
\with \(\Gamma \vdash e_1 : T_1\) \show \(?thesis\) by \(\text{(simp add: typeBinOp3)}\)
\qed

\textbf{lemma \textit{exprTypingLow}}:
\assumes \(\Gamma \vdash e : \text{Low}\) and \(x \in \text{fv} e\) \shows \(\Gamma \vdash x = \text{Some Low}\)

\textbf{end}
using assms

proof (induct e)
  case (Val V)
  have \( \text{fv}(\text{Val } V) = \{\} \) by (rule FVc)
  with \( x \in \text{fv}(\text{Val } V) \) have False by auto
  thus ?thesis by simp
next
  case (Var V)
  from \( x \in \text{fv}(\text{Var } V) \) have \( xV \colon x = V \) by simp
  from \( \Gamma \vdash \text{Var } V : \text{Low} \) have \( \Gamma V = \text{Some } \text{Low} \) by (auto elim:secExprTyping.cases)
  thus \( \text{thesis} \) by simp
next
  case (BinOp e1 bop e2)
  note IH1 = \( \bigwedge T . \Gamma \vdash e1 : T \implies \text{fv} e1 \subseteq \text{dom } \Gamma \)\)
  note IH2 = \( \bigwedge T . \Gamma \vdash e2 : T \implies \text{fv} e2 \subseteq \text{dom } \Gamma \)\)
  from \( \Gamma \vdash e1 \ll bop \gg e2 : \text{Low} \) have \( \Gamma \vdash e1 : \text{Low} \) and \( \Gamma \vdash e2 : \text{Low} \)
  by (auto elim:secExprTyping.cases)
  from \( x \in \text{fv}(e1 \ll bop \gg e2) \) have \( x \in \text{fv} e1 \cup \text{fv} e2 \) by (simp add: FVe)
  hence \( x \in \text{fv} e1 \lor x \in \text{fv} e2 \) by auto
  thus ?case
  proof
    assume \( x \in \text{fv} e1 \)
    with IH1[OF \( \Gamma \vdash e1 : \text{Low} \)] show \( \text{thesis} \) by auto
  next
    assume \( x \in \text{fv} e2 \)
    with IH2[OF \( \Gamma \vdash e2 : \text{Low} \)] show \( \text{thesis} \) by auto
  qed
qed

lemma typeableFreevars:
  assumes \( \Gamma \vdash e : T \) shows \( \text{fv} e \subseteq \text{dom } \Gamma \)
using assms

proof (induct e arbitrary: T)
  case (Val V)
  have \( \text{fv}(\text{Val } V) = \{\} \) by (rule FVc)
  thus ?case by simp
next
  case (Var V)
  show ?case
  proof
    fix \( x \) assume \( x \in \text{fv}(\text{Var } V) \)
    hence \( x = V \) by simp
    from \( \Gamma \vdash \text{Var } V : T \) have \( \Gamma V = \text{Some } T \) by (auto elim:secExprTyping.cases)
    with \( x = V \) show \( x \in \text{dom } \Gamma \) by auto
  qed
next
  case (BinOp e1 bop e2)
  note IH1 = \( \forall T . \Gamma \vdash e1 : T \implies \text{fv} e1 \subseteq \text{dom } \Gamma \)
note \( IH2 = (\forall T. \Gamma \vdash e_2 : T \implies \text{fv} e_2 \subseteq \text{dom} \Gamma) \)
show \(?\text{case}\)
proof
  fix \( x \) assume \( x \in \text{fv} (e_1 <\text{bop}> e_2) \)
  from \( \Gamma \vdash e_1 <\text{bop}> e_2 : T \)
  have \( Q; (\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2) \)
    by (rule exprBinopTypeable)
then obtain \( T_1 \) where \( \Gamma \vdash e_1 : T_1 \) by blast
from \( Q \) obtain \( T_2 \) where \( \Gamma \vdash e_2 : T_2 \) by blast
from \( IH1[\text{OF} \ (\exists T_1. \Gamma \vdash e_1 : T_1)] \) have \( \text{fv} e_1 \subseteq \text{dom} \Gamma \).
moreover
from \( IH2[\text{OF} \ (\exists T_2. \Gamma \vdash e_2 : T_2)] \) have \( \text{fv} e_2 \subseteq \text{dom} \Gamma \).
ultimately have \( \text{fv} e_1 \cup \text{fv} e_2 \subseteq \text{dom} \Gamma \) by auto
hence \( \text{fv} (e_1 <\text{bop}> e_2) \subseteq \text{dom} \Gamma \) by (simp add: FVc)
with \( x \in \text{fv} (e_1 <\text{bop}> e_2) \) show \( x \in \text{dom} \Gamma \) by auto
qed
qed

lemma \( \text{exprNotNone} \):
assumes \( \Gamma \vdash e : T \) and \( \text{fv} e \subseteq \text{dom} s \)
shows \([e] s \neq \text{None}\)
using assms
proof (induct \( e \) arbitrary: \( \Gamma \ T s \))
  case (Val \( v \))
  show \(?\text{case}\) by (simp add: Val)
next
  case (Var \( V \))
  have \([\text{Var} V] s = s \ V\) by (simp add: Var)
  have \( V \in \text{fv} (\text{Var} V)\) by (auto simp add: FVv)
with \( \text{fv} (\text{Var} V) \subseteq \text{dom} s\) have \( V \in \text{dom} s \) by simp
thus \(?\text{case}\) by auto
next
  case (BinOp \( e_1 \ bop\ e_2 \))
  note \( IH1 = (\forall T. \Gamma \vdash e_1 : T; \text{fv} e_1 \subseteq \text{dom} s \implies \[e1] s \neq \text{None}) \)
  note \( IH2 = (\forall T. \Gamma \vdash e_2 : T; \text{fv} e_2 \subseteq \text{dom} s \implies \[e2] s \neq \text{None}) \)
from \( \Gamma \vdash e_1 <\text{bop}> e_2 : T \) have \( (\exists T_1. \Gamma \vdash e_1 : T_1) \land (\exists T_2. \Gamma \vdash e_2 : T_2) \)
  by (rule exprBinopTypeable)
then obtain \( T_1 \ T_2 \) where \( \Gamma \vdash e_1 : T_1 \) and \( \Gamma \vdash e_2 : T_2 \) by blast
from \( \text{fv} (e_1 <\text{bop}> e_2) \subseteq \text{dom} s\) have \( \text{fv} e_1 \cup \text{fv} e_2 \subseteq \text{dom} s \) by (simp add: FVc)
  hence \( \text{fv} e_1 \subseteq \text{dom} s \) and \( \text{fv} e_2 \subseteq \text{dom} s \) by auto
from \( IH1[\text{OF} \ (\exists T_1. \Gamma \vdash e_1 : T_1; \text{fv} e_1 \subseteq \text{dom} s)] \) have \([e1] s \neq \text{None} \).
moreover from \( IH2[\text{OF} \ (\exists T_2. \Gamma \vdash e_2 : T_2; \text{fv} e_2 \subseteq \text{dom} s)] \) have \([e2] s \neq \text{None} \).
ultimately show \(?\text{case}\)
  apply (cases \( \text{bop} \)) apply auto
  apply (case_tac \( y \), auto, case_tac \( ya \), auto)+
done
2.3 Noninterference definitions

2.3.1 Low Equivalence

Low Equivalence is reflexive even if the involved states are undefined. But in non-reflexive situations low variables must be initialized (i.e. ∈ dom state), otherwise the proof will not work. This is not a restriction, but a natural requirement, and could be formalized as part of a standard type system.

Low equivalence is also symmetric and transitive (see lemmas) hence an equivalence relation.

definition lowEquiv :: typeEnv ⇒ state ⇒ state ⇒ bool (Γ ⊢ s1 ≈_L s2)
where
Γ ⊢ s1 ≈_L s2 ≡ ∀ v ∈ dom Γ. Γ v = Some Low → (s1 v = s2 v)

lemma lowEquivReflexive: Γ ⊢ s1 ≈_L s1
by (simp add: lowEquiv-def)

lemma lowEquivSymmetric: Γ ⊢ s1 ≈_L s2 =⇒ Γ ⊢ s2 ≈_L s1
by (simp add: lowEquiv-def)

lemma lowEquivTransitive: [[Γ ⊢ s1 ≈_L s2; Γ ⊢ s2 ≈_L s3]] =⇒ Γ ⊢ s1 ≈_L s3
by (simp add: lowEquiv-def)

2.3.2 Non Interference

definition nonInterference :: typeEnv ⇒ com ⇒ bool
where nonInterference Γ c ≡
(∀ s1 s2 s1' s2'. Γ ⊢ s1 ≈_L s2 ∧ ⟨c, s1⟩ →∗ ⟨Skip, s1'⟩ ∧ ⟨c, s2⟩ →∗ ⟨Skip, s2'⟩) →
Γ ⊢ s1' ≈_L s2'

lemma nonInterferenceI:
[[∀ s1 s2 s1' s2'. [Γ ⊢ s1 ≈_L s2; ⟨c, s1⟩ →∗ ⟨Skip, s1'⟩; ⟨c, s2⟩ →∗ ⟨Skip, s2'⟩]] →
Γ ⊢ s1' ≈_L s2'] =⇒ nonInterference Γ c
by (auto simp: nonInterference-def)

lemma interpretLow:
assumes Γ ⊢ s1 ≈_L s2 and all: ∀ V ∈ fv e. Γ V = Some Low
shows [e] s1 = [e] s2
using all
proof (induct e)
case (Val v)
  show ?case by (simp add: Val)
next
case (Var V)
have \([\text{Var} \ V] \ s1 = s1 \ V \) and \([\text{Var} \ V] \ s2 = s2 \ V\) by(auto simp:Var)
have \(V \in \text{fv} \ (\text{Var} \ V)\) by(simp add:FVv)
from \((V \in \text{fv} \ (\text{Var} \ V)) \) \(\forall X \in \text{fv} \ (\text{Var} \ V)\). \(X = \text{Some Low}\) have \(\Gamma \ V = \text{Some Low}\) by simp
with \(\text{assms}\) have \(s1 = s1\) \(V\) by(auto simp add:lowEquiv-def)
thus \(?\text{case}\) by \(\text{auto}\)
next
case (BinOp \(e1 \ bop \ e2\))
note \(\text{IH1} = (\forall V \in \text{fv} \ e1). \Gamma \ V = \text{Some Low} \Longrightarrow \) \([e1] s1 = [e1] s2\)
note \(\text{IH2} = (\forall V \in \text{fv} \ e2). \Gamma \ V = \text{Some Low} \Longrightarrow \) \([e2] s1 = [e2] s2\)
from \(\forall V \in \text{fv} \ (e1 <bop> e2)\). \(\Gamma \ V = \text{Some Low}\) have \(\forall V \in \text{fv} \ e1. \Gamma \ V = \text{Some Low}\) and \(\forall V \in \text{fv} \ e2. \Gamma \ V = \text{Some Low}\) by \(\text{auto}\)
moreover
from \(\text{IH2}[\text{OF} \ (\forall V \in \text{fv} \ e2) = \text{Some Low}]\) have \([e2] s1 = [e2] s2\)
ultimately show \(?\text{case}\) by(\(\text{cases} \ [e1] \ s2, \text{auto}\))
qed

**Lemma** interpretLow2:
assumes \(\Gamma \vdash e : \text{Low} \\) and \(\Gamma \vdash s1 \approx_L s2\) shows \([e] s1 = [e] s2\)
proof –
from \(\Gamma \vdash e : \text{Low}\) have \(\text{fv} \ e \subseteq \text{dom} \ \Gamma\) by(auto dest:typeableFreevars)
have \(\forall x \in \text{fv} \ e. \Gamma \ x = \text{Some Low}\)
proof
fix \(x\) assume \(x \in \text{fv} \ e\)
with \(\Gamma \vdash e : \text{Low}\) show \(\Gamma \ x = \text{Some Low}\) by(auto intro:exprTypingLow)
qed
with \(\Gamma \vdash s1 \approx_L s2\) show \(?\text{thesis}\) by(\(\text{rule} \ \text{interpretLow}\))
qed

**Lemma** assignNIhighlemma:
assumes \(\Gamma \vdash s1 \approx_L s2\) and \(\Gamma \ V = \text{Some High}\) and \(s1' = s1(V := [e] s1)\)
and \(s2' = s2(V := [e] s2)\)
shows \(\Gamma \vdash s1' \approx_L s2'\)
proof
\{ fix \(V'\) assume \(V' \in \text{dom} \ \Gamma\) and \(\Gamma \ V' = \text{Some Low}\)
from \(\Gamma \vdash s1 \approx_L s2\)
\(\Gamma \ V' = \text{Some Low}\) have \(s1 V' = s2 V'\)
by(auto simp add:lowEquiv-def)
have \(s1' V' = s2' V'\)
proof(cases \(V' = V\))
case True
with \(\Gamma \ V' = \text{Some Low}\) \(\Gamma \ V = \text{Some High}\) have \(?\text{False}\) by simp
thus \(?\text{thesis}\) by simp
next
case False
with \(s1' = s1(V := [e] s1)\) \(s2' = s2(V := [e] s2)\)
have \(s_1 V' = s_1' V'\) and \(s_2 V' = s_2' V'\) by auto

with \(s_1 V' = s_2 V'\) show \(?\)thesis by simp

qed

\}

\thus \(?\)thesis by(\(auto\) simp add:lowEquiv-def)

qed

\begin{lemma}
\textit{assignNIlowlemma:}
\end{lemma}

\begin{proof}
\begin{enumerate}
\item \textbf{fix} \(V'\) \textbf{assume} \(V' \in \text{dom} \Gamma\) and \(\Gamma V' = \text{Some Low}\)
\item \textbf{from} \(\Gamma \vdash s_1 \approx_{L} s_2\) \(\Gamma V' = \text{Some Low}\)
\item \textbf{have} \(s_1 V' = s_2 V'\) by(\(auto\) simp add:lowEquiv-def)
\item \textbf{have} \(s_1' V' = s_2' V'\)
\end{enumerate}

\begin{proof}
\begin{enumerate}
\item \textbf{cases} \(V' = V\)
\item \textbf{case} \(\text{True}\)
\item \textbf{with} \(s_1' = s_1(V := [e] s_1)\) \(s_2' = s_2(V := [e] s_2)\)
\item \textbf{have} \(s_1' V' = [e] s_1\) and \(s_2' V' = [e] s_2\) by \(auto\)
\item \textbf{by}(\(auto\) intro:interpretLow2)
\item \textbf{with} \(s_1' V' = [e] s_1\) \(s_2' V' = [e] s_2\) show \(?\)thesis by simp
\end{enumerate}

\begin{proof}
\begin{enumerate}
\item \textbf{next}
\item \textbf{case} \(\text{False}\)
\item \textbf{with} \(s_1' = s_1(V := [e] s_1)\) \(s_2' = s_2(V := [e] s_2)\)
\item \textbf{have} \(s_1' V' = s_1 V'\) and \(s_2' V' = s_2 V'\) by \(auto\)
\item \textbf{with} \(\text{False}\) \(s_1' V' = s_2 V'\) \(s_2' V' = s_2 V'\)
\item \textbf{show} \(?\)thesis by \(auto\)
\end{enumerate}

\end{proof}

\end{proof}

\end{proof}

\thus \(?\)thesis by(simp add:lowEquiv-def)

\end{proof}

Sequential Compositionality is given the status of a theorem because compositionality is no longer valid in case of concurrency

\begin{theorem}
\textit{SeqCompositionality:}
\end{theorem}

\begin{proof}
\begin{enumerate}
\item \textbf{fix} \(s_1 s_2 s_1' s_2'\)
\item \textbf{assume} \(\Gamma \vdash s_1 \approx_{L} s_2\) and \(\langle c_1;c_2,s_1\rangle \rightarrow* \langle \text{Skip},s_1' \rangle\)
\item \textbf{and} \(\langle c_1;c_2,s_2\rangle \rightarrow* \langle \text{Skip},s_2' \rangle\)
\end{enumerate}

\begin{from}
\begin{enumerate}
\item \textbf{obtain} \(s_1''\) \(\text{where} \langle c_1,s_1\rangle \rightarrow* \langle \text{Skip},s_1'' \rangle\)
\item \(\langle c_2,s_1'' \rangle \rightarrow* \langle \text{Skip},s_1' \rangle\) by(\(auto\) dest:Seq-reds)
\item \(\langle c_1;c_2,s_2\rangle \rightarrow* \langle \text{Skip},s_2' \rangle\) \(\text{obtain} s_2''\) \(\text{where} \langle c_1,s_2\rangle \rightarrow* \langle \text{Skip},s_2'' \rangle\)
\end{enumerate}

\end{from}

\end{proof
and \((c2, s2')\) \(\rightarrow\) \((\text{Skip}, s2')\) \texttt{by(auto dest:Seg-reds)}

from \(\Gamma \vdash s1 \approxL s2\) \(\rightarrow\) \((\text{Skip}, s1')\) \((c1, s1) \rightarrow\) \((c1, s1) \rightarrow\) \((\text{Skip}, s2')\)

\texttt{nonInterference \Gamma c1}

\texttt{have} \(\Gamma \vdash s1'' \approxL s2''\) \texttt{by(auto simp:nonInterference-def)}

\texttt{with} \((c2, s1''\) \((\text{Skip}, s1'\) \((c2, s2'\) \((\text{Skip}, s2'\) \((\text{nonInterference} \Gamma c2\)

\texttt{show} \(\Gamma \vdash s1' \approxL s2'\) \texttt{by(auto simp:nonInterference-def)}

\texttt{qed}

\textbf{lemma} \texttt{WhileStepInduct:}

\texttt{assumes} \texttt{while:\langle while (b) c,s1 \rightarrow\ (\text{Skip}, s2)\}

\texttt{and body:\langle s2 . (c,s1) \rightarrow\ (\text{Skip}, s2)\ \rightarrow\ (\text{Skip}, s2) \rightarrow\ (\text{Skip}, s2) \rightarrow\ (\text{nonInterference} \Gamma c2\)

\texttt{show} \(\Gamma \vdash s1 \approxL s2\)

\texttt{using} \texttt{while}

\texttt{proof (induct rule:while-reds-induct)}

\texttt{case (false s) thus \(?case by(auto simp add:lowEquiv-def)\}

\texttt{next}

\texttt{case (true s1 s2)}

\texttt{from body\langle OF \langle c,s1 \rightarrow\ (\text{Skip}, s3)\rangle\ have} \(\Gamma \vdash s1 \approxL s3\) \texttt{by simp}

\texttt{with} \(\Gamma \vdash s3 \approxL s2\) \texttt{show} \(\?case by(auto intro:lowEquivTransitive)\)

\texttt{qed}

In case control conditions from if/while are high, the body of an if/while must not change low variables in order to prevent implicit flow. That is, start and end state of any if/while body must be low equivalent.

\textbf{theorem} \texttt{highBodies:}

\texttt{assumes} \texttt{\Gamma,High \vdash c and (c,s1) \rightarrow\ (\text{Skip}, s2)\}

\texttt{shows} \(\Gamma \vdash s1 \approxL s2\)

— all intermediate states must be well formed, otherwise the proof does not work for uninitialized variables. Thus it is propagated through the theorem conclusion

\texttt{using asms}

\texttt{proof(induct c arbitrary:s1 s2 rule:com.induct)}

\texttt{case Skip}

\texttt{from \langle (\text{Skip}, s1) \rightarrow\ (\text{Skip}, s2)\ have s1 = s2 by (auto dest:Skip-reds)\}

\texttt{thus \(?case by(simp add:lowEquiv-def)\}

\texttt{next}

\texttt{case (LAss \ V e)}

\texttt{from \Gamma,High \vdash V:=e} \texttt{have} \(\Gamma \ V = \text{Some High by(auto elim:secComTyping.cases)\}

\texttt{from \langle V:=e,s1 \rightarrow\ (\text{Skip}, s2)\ have s2 = s1(V:= [e]s1) by (auto intro:LAss-reds)\}

\{ \texttt{fix \ V' assume \ V' \in dom \Gamma and} \ \Gamma \ V' = \text{Some Low}\}

\texttt{have s1 V' = s2 V'}

\texttt{proof(cases V' = V)\}

\texttt{case True}

\texttt{with} \(\Gamma \ V' = \text{Some Low}\) \(\Gamma \ V = \text{Some High}\) \texttt{have False by simp}

\texttt{thus \(?thesis by simp\}

\texttt{next}

\texttt{case False}

\texttt{with} \(s2 = s1(V:= [e]s1)\) \texttt{show} \(\?thesis by simp\}
qed
}
thus \(\text{case by (auto simp add: lowEquiv-def)}\)

next
case (Seq \(c_1\ c_2\))
note IH1 = \(\lambda s_1 s_2. \; [\Gamma, \text{High} \vdash \Gamma \vdash s_1 \approx_L s_2] \Rightarrow \Gamma \vdash s_1 \approx_L s_2\)
note IH2 = \(\lambda s_1 s_2. \; [\Gamma, \text{High} \vdash \Gamma \vdash s_1 \approx_L s_2] \Rightarrow \Gamma \vdash s_1 \approx_L s_2\)
from \(\Gamma, \text{High} \vdash c_1::c_2\) have \(\Gamma, \text{High} \vdash c_1\) and \(\Gamma, \text{High} \vdash c_2\)
by (auto elim: secComTyping.cases)
from \((\langle c_1::c_2, s_1\rangle \rightarrow \langle \text{Skip}, s_2\rangle)\)
have \(\exists s_3. \; \langle c_1, s_1\rangle \rightarrow \langle \text{Skip}, s_3\rangle \wedge \langle c_2, s_3\rangle \rightarrow \langle \text{Skip}, s_2\rangle\) by (auto intro: Seq-reds)
then obtain \(s_3\) where \((c_1, s_1) \rightarrow \langle \text{Skip}, s_3\rangle\) and \((c_2, s_3) \rightarrow \langle \text{Skip}, s_2\rangle\) by auto
from IH1[OF \(\Gamma, \text{High} \vdash c_1\langle c_1, s_1\rangle \rightarrow \langle \text{Skip}, s_3\rangle]\]
have \(\Gamma \vdash s_1 \approx_L s_3\) by simp
from IH2[OF \(\Gamma, \text{High} \vdash c_2\langle c_2, s_3\rangle \rightarrow \langle \text{Skip}, s_2\rangle]\]
have \(\Gamma \vdash s_3 \approx_L s_2\) by simp
from \(\Gamma \vdash s_1 \approx_L s_3\); \(\Gamma \vdash s_3 \approx_L s_2\) show \(\text{case}\)
by (auto intro: lowEquivTransitive)

next
case (Cond \(b\ c_1\ c_2\))
note IH1 = \(\lambda s_1 s_2. \; [\Gamma, \text{High} \vdash c_1\langle c_1, s_1\rangle \rightarrow \langle \text{Skip}, s_2\rangle] \Rightarrow \Gamma \vdash s_1 \approx_L s_2\)
note IH2 = \(\lambda s_1 s_2. \; [\Gamma, \text{High} \vdash c_2\langle c_2, s_1\rangle \rightarrow \langle \text{Skip}, s_2\rangle] \Rightarrow \Gamma \vdash s_1 \approx_L s_2\)
from \(\Gamma, \text{High} \vdash \text{if } (b) \text{ c1 else c2}\) have \(\Gamma, \text{High} \vdash c_1\) and \(\Gamma, \text{High} \vdash c_2\)
by (auto elim: secComTyping.cases)
from \((\langle \text{if } (b) \text{ c1 else c2}, s_1\rangle \rightarrow \langle \text{Skip}, s_2\rangle)\)
have \([b] s_1 = \text{Some true} \vee [b] s_1 = \text{Some false}\) by (auto dest: Cond-True-or-False)
thus \(\text{case}\)
proof
assume \([b] s_1 = \text{Some true}\)
with \((\langle \text{if } (b) \text{ c1 else c2}, s_1\rangle \rightarrow \langle \text{Skip}, s_2\rangle)\) have \(\langle c_1, s_1\rangle \rightarrow \langle \text{Skip}, s_2\rangle\)
by (auto intro: CondTrue-reds)
from IH1[OF \(\Gamma, \text{High} \vdash c_1\langle c_1, s_1\rangle\); \(\text{this}\)] show \(\text{thesis}\).
next
assume \([b] s_1 = \text{Some false}\)
with \((\langle \text{if } (b) \text{ c1 else c2}, s_1\rangle \rightarrow \langle \text{Skip}, s_2\rangle)\) have \(\langle c_2, s_1\rangle \rightarrow \langle \text{Skip}, s_2\rangle\)
by (auto intro: CondFalse-reds)
from IH2[OF \(\Gamma, \text{High} \vdash c_2\langle c_2, s_1\rangle\); \(\text{this}\)] show \(\text{thesis}\).
qed

defined

next
case (While \(b\ c')
note IH = \(\lambda s_1 s_2. \; [\Gamma, \text{High} \vdash c'\langle c', s_1\rangle \rightarrow \langle \text{Skip}, s_2\rangle] \Rightarrow \Gamma \vdash s_1 \approx_L s_2\)
from \(\Gamma, \text{High} \vdash \text{while } (b) \text{ c'}\) have \(\Gamma, \text{High} \vdash c'\) by (auto elim: secComTyping.cases)
from IH[OF \(\text{this}\)]
have \(\langle s_1 s_2. \; [\langle c', s_1\rangle \rightarrow \langle \text{Skip}, s_2\rangle\] \Rightarrow \Gamma \vdash s_1 \approx_L s_2\).
with \((\langle \text{while } (b) \text{ c'}, s_1\rangle \rightarrow \langle \text{Skip}, s_2\rangle)\) \(\Gamma, \text{High} \vdash c'\)
show \(\text{case by (auto dest: WhileStepInduct)\)
qed
**lemma** CondHighCompositionality:

assumes \( \Gamma, \text{High} \vdash \text{if} \ (b) \ c1 \text{ else } c2 \)

shows nonInterference \( \Gamma \ (\text{if} \ (b) \ c1 \text{ else } c2) \)

**proof** (rule nonInterferenceI)

- fix \( s1 \ s2 \ s1' \ s2' \)
- assume \( \Gamma \vdash s1 \approx_L s2 \text{ and } (\text{if} \ (b) \ c1 \text{ else } c2,s1) \rightarrow^* \langle \text{Skip},s1' \rangle \)
- and \( (\text{if} \ (b) \ c1 \text{ else } c2,s2) \rightarrow^* \langle \text{Skip},s2' \rangle \)
- show \( \Gamma \vdash s1' \approx_L s2' \)

**proof**

- from \( \Gamma, \text{High} \vdash (\text{if} \ (b) \ c1 \text{ else } c2) \langle (\text{if} \ (b) \ c1 \text{ else } c2,s1) \rightarrow^* \langle \text{Skip},s1' \rangle \rangle \)
- have \( \Gamma \vdash s1 \approx_L s1' \text{ by (auto dest:highBodies)} \)
- from \( \Gamma, \text{High} \vdash (\text{if} \ (b) \ c1 \text{ else } c2) \langle (\text{if} \ (b) \ c1 \text{ else } c2,s2) \rightarrow^* \langle \text{Skip},s2' \rangle \rangle \)
- have \( \Gamma \vdash s2 \approx_L s2' \text{ by (auto dest:highBodies)} \)
- with \( \Gamma \vdash s1 \approx_L s2 \) have \( \Gamma \vdash s1 \approx_L s2' \text{ by (auto intro:lowEquivTransitive)} \)
- from \( \Gamma \vdash s1 \approx_L s1' \) have \( \Gamma \vdash s1' \approx_L s1 \text{ by (auto intro:lowEquivSymmetric)} \)
- with \( \Gamma \vdash s1 \approx_L s2' \) show \( \langle \text{thesis} \rangle \text{ by (auto intro:lowEquivTransitive)} \)

qed

**lemma** CondLowCompositionality:

assumes nonInterference \( \Gamma \ c1 \text{ and } nonInterference \ \Gamma \ c2 \text{ and } \Gamma \vdash b : \text{Low} \)

shows nonInterference \( \Gamma \ (\text{if} \ (b) \ c1 \text{ else } c2) \)

**proof** (rule nonInterferenceI)

- fix \( s1 \ s2 \ s1' \ s2' \)
- assume \( \Gamma \vdash s1 \approx_L s2 \text{ and } (\text{if} \ (b) \ c1 \text{ else } c2,s1) \rightarrow^* \langle \text{Skip},s1' \rangle \)
- and \( (\text{if} \ (b) \ c1 \text{ else } c2,s2) \rightarrow^* \langle \text{Skip},s2' \rangle \)
- from \( \langle (\text{if} \ (b) \ c1 \text{ else } c2,s1) \rightarrow^* \langle \text{Skip},s1' \rangle \rangle \)
- have \( \langle b \rangle \ s1 = \text{Some true} \text{ by (auto intro:CondTrue-or-False)} \)
- thus \( \Gamma \vdash s1' \approx_L s2' \)

**proof**

- assume \( \langle b \rangle \ s1 = \text{Some true} \)
  - with \( \langle b \rangle \ s1 = \text{Some false} \text{ by (auto intro:CondTrue-reds)} \)
  - from \( \langle b \rangle \ s1 = \text{Some true} \) have \( \langle (\text{if} \ (b) \ c1 \text{ else } c2,s1) \rightarrow^* \langle \text{Skip},s1' \rangle \⟩ \)
  - have \( \langle c1,s1 \rangle \rightarrow^* \langle \text{Skip},s1' \rangle \text{ by (auto intro:CondTrue-reds)} \)
  - from \( \langle b \rangle \ s2 = \text{Some true} \) have \( \langle (\text{if} \ (b) \ c1 \text{ else } c2,s2) \rightarrow^* \langle \text{Skip},s2' \rangle \⟩ \)
  - have \( \langle c1,s2 \rangle \rightarrow^* \langle \text{Skip},s2' \rangle \text{ by (auto intro:CondTrue-reds)} \)
  - with \( \langle c1,s1 \rangle \rightarrow^* \langle \text{Skip},s1' \rangle \) show \( \langle \text{thesis} \rangle \text{ by (auto simp:nonInterference-def)} \)

**next**

- assume \( \langle b \rangle \ s1 = \text{Some false} \)
  - with \( \langle b \rangle \ s1 = \text{Some false} \text{ by (auto intro:CondTrue-reds)} \)
  - from \( \langle b \rangle \ s1 = \text{Some false} \) have \( \langle (\text{if} \ (b) \ c1 \text{ else } c2,s1) \rightarrow^* \langle \text{Skip},s1' \rangle \⟩ \)
  - have \( \langle c2,s1 \rangle \rightarrow^* \langle \text{Skip},s1' \rangle \text{ by (auto intro:CondFalse-reds)} \)
  - from \( \langle b \rangle \ s2 = \text{Some false} \) have \( \langle (\text{if} \ (b) \ c1 \text{ else } c2,s2) \rightarrow^* \langle \text{Skip},s2' \rangle \⟩ \)
have \((c_2, s_2) \rightarrow^* (\text{Skip}, s_2')\) by \((\text{auto intro:CondFalse-reds})\)
with \((c_2, s_1) \rightarrow^* (\text{Skip}, s_1')\); \(\Gamma \vdash s_1 \approx_L s_2\); \(\text{nonInterference} \Gamma c_2\)
show \(?\text{thesis}\) by \((\text{auto simp:nonInterference-def})\)
qed

lemma \(\text{WhileHighCompositionality}\):
assumes \(\Gamma, \text{High} \vdash \text{while} (b) c'\)
shows \(\text{nonInterference} \Gamma (\text{while} (b) c')\)
proof \((\text{rule nonInterferenceI})\)
fix \(s_1, s_2, s_1', s_2'\)
assume \(\Gamma \vdash s_1 \approx_L s_2\) and \(\{\text{while} (b) c', s_1\} \rightarrow^* (\text{Skip}, s_1')\)
and \(\{\text{while} (b) c', s_2\} \rightarrow^* (\text{Skip}, s_2')\)
show \(\Gamma \vdash s_1' \approx_L s_2'\)
proof
from \(\Gamma, \text{High} \vdash \text{while} (b) c'\); \{\text{while} (b) c', s_1\} \rightarrow^* (\text{Skip}, s_1')\)
have \(\Gamma \vdash s_1 \approx_L s_1'\) by \((\text{auto dest:highBodies})\)
from \(\Gamma, \text{High} \vdash \text{while} (b) c'\); \{\text{while} (b) c', s_2\} \rightarrow^* (\text{Skip}, s_2')\)
have \(\Gamma \vdash s_2 \approx_L s_2'\) by \((\text{auto dest:highBodies})\)
with \(\Gamma \vdash s_1 \approx_L s_2\) have \(\Gamma \vdash s_1 \approx_L s_2\) by \((\text{auto intro:lowEquivTransitive})\)
from \(\Gamma \vdash s_1 \approx_L s_1'\) have \(\Gamma \vdash s_1 \approx_L s_1'\) by \((\text{auto intro:lowEquivSymmetric})\)
with \(\Gamma \vdash s_1 \approx_L s_2'\) show \(?\text{thesis}\) by \((\text{auto intro:lowEquivTransitive})\)
qed

lemma \(\text{WhileLowStepInduct}\):
assumes \(\text{while1}: \{\text{while} (b) c', s_1\} \rightarrow^* (\text{Skip}, s_1')\)
and \(\text{while2}: \{\text{while} (b) c', s_2\} \rightarrow^* (\text{Skip}, s_2')\)
and \(\Gamma \vdash b : \text{Low}\)
and \(\text{body}: \{s_1, s_1' \approx_L s_2, \Gamma \vdash c', s_1 \rightarrow^* (\text{Skip}, s_1'); c', s_2 \rightarrow^* (\text{Skip}, s_2');\}
\(\Gamma \vdash s_1 \approx_L s_2\) \implies \(\Gamma \vdash s_1' \approx_L s_2'\)
and \(\text{le}: \Gamma \vdash s_1 \approx_L s_2\)
shows \(\Gamma \vdash s_1' \approx_L s_2'\)
using \(\text{while1 le while2}\)
proof \((\text{induct arbitrary:s2 rule:while-reds-induct})\)
case \((\text{false s1})\)
from \(\Gamma \vdash b : \text{Low}\); \(\Gamma \vdash s_1 \approx_L s_2\) have \([b] s_1 = [b] s_2\) by \((\text{auto intro:interpretLow2})\)
with \(([b] s_1 = \text{Some false})\) have \([b] s_2 = \text{Some false}\) by \(\text{simp}\)
with \(\{\text{while} (b) c', s_2\} \rightarrow^* (\text{Skip}, s_2')\) have \(s_2 = s_2'\) by \((\text{auto intro:WhileFalse-reds})\)
with \(\Gamma \vdash s_1 \approx_L s_2\) show \(?\text{case}\) by \(\text{auto}\)
next
case \((\text{true s1 s1''})\)
note \(IH = \{s_2''\}; [\Gamma \vdash s_1'' \approx_L s_2'']; \{\text{while} (b) c', s_2''\} \rightarrow^* (\text{Skip}, s_2'')\)
\implies \(\Gamma \vdash s_1' \approx_L s_2'\)
from \(\Gamma \vdash b : \text{Low}\); \(\Gamma \vdash s_1 \approx_L s_2\) have \([b] s_1 = [b] s_2\) by \((\text{auto intro:interpretLow2})\)
with \(([b] s_1 = \text{Some true})\) have \([b] s_2 = \text{Some true}\) by \(\text{simp}\)
with \((\text{while} (b) \ c',s2) \mapsto \langle \text{Skip},s2' \rangle\) obtain \(s2''\) where \((c',s2) \mapsto \langle \text{Skip},s2' \rangle\) and \((\text{while} (b) \ c',s2') \mapsto \langle \text{Skip},s2' \rangle\) by (auto dest:WhileTrue-refs)

from body[OF \((c',s1) \mapsto \langle \text{Skip},s1' \rangle; \langle c',s2 \rangle \mapsto \langle \text{Skip},s2' \rangle\); \(\Gamma \vdash s1 \approx_L s2\)]

have \(\Gamma \vdash s1'' \approx_L s2''\).

from IH[OF this \((\text{while} (b) \ c',s2') \mapsto \langle \text{Skip},s2' \rangle\)] show \(?case\).

qed

lemma WhileLowCompositionality:
assumes \(\text{nonInterference} \ \Gamma \ c' \ \text{and} \ \Gamma \vdash b : \text{Low} \ \text{and} \ \Gamma',\text{Low} \vdash c'\)
shows \(\text{nonInterference} \ \Gamma \ \langle \text{while} (b) \ c' \rangle\)

proof (rule nonInterferenceI)

fix \(s1 \ s2 \ s1' \ s2'\)

assume \(\Gamma \vdash s1 \approx_L s2\) and \((\text{while} (b) \ c',s1) \mapsto \langle \text{Skip},s1' \rangle\) and \((\text{while} (b) \ c',s2) \mapsto \langle \text{Skip},s2' \rangle\)

\{ fix \(s1 \ s2 \ s1'' \ s2''\)

assume \(c',s1) \mapsto \langle \text{Skip},s1' \rangle\) and \((c',s2) \mapsto \langle \text{Skip},s2' \rangle\) and \(\Gamma \vdash s1 \approx_L s2\)

with \(\text{nonInterference} \ c'\) have \(\Gamma \vdash s1'' \approx_L s2''\)

by (auto simp:nonInterference-def)
\}

hence \(\bigwedge s1 \ s2 \ s1'' \ s2''\). \([\langle c',s1) \mapsto \langle \text{Skip},s1' \rangle; \langle c',s2 \rangle \mapsto \langle \text{Skip},s2' \rangle\];

\(\Gamma \vdash s1 \approx_L s2\) \(\Longrightarrow\) \(\Gamma \vdash s1'' \approx_L s2''\) by auto

with \((\text{while} (b) \ c',s1) \mapsto \langle \text{Skip},s1' \rangle; \langle \text{while} (b) \ c',s2 \rangle \mapsto \langle \text{Skip},s2' \rangle\)

\(\Gamma \vdash b : \text{Low}\) \(\Gamma \vdash s1 \approx_L s2\) show \(\Gamma \vdash s1' \approx_L s2'\)

by (auto intro:WhileLowStepInduct)

qed

and now: the main theorem:

theorem seeTypeImpliesNonInterference:

\(\Gamma,T \vdash c \Longrightarrow \text{nonInterference} \ \Gamma \ c\)

proof (induct \(c\) arbitrary: \(T\) rule:com.induct)

case \(\text{Skip}\)

show \(?case\)

proof (rule nonInterferenceI)

fix \(s1 \ s2 \ s1' \ s2'\)

assume \(\Gamma \vdash s1 \approx_L s2\) and \(\langle \text{Skip},s1 \rangle \mapsto \langle \text{Skip},s1' \rangle\) and \(\langle \text{Skip},s2 \rangle \mapsto \langle \text{Skip},s2' \rangle\)

from \(\langle \text{Skip},s1 \rangle \mapsto \langle \text{Skip},s1' \rangle\) have \(s1 = s1'\) by (auto dest:Skip-refs)

from \(\langle \text{Skip},s2 \rangle \mapsto \langle \text{Skip},s2' \rangle\) have \(s2 = s2'\) by (auto dest:Skip-refs)

from \(\Gamma \vdash s1 \approx_L s2\) and \(\langle s1 = s1'\) and \(s2 = s2'\)

show \(\Gamma \vdash s1' \approx_L s2'\) by simp

qed

next

case \(LA\ss \ V \ e\)

from \(\Gamma,T \vdash V := e\)

have \(\text{varprem}:(\Gamma \ V = \text{Some High}) \lor (\Gamma \ V = \text{Some Low} \ \land \ \Gamma \vdash e : \text{Low} \ \land \ T = \text{Low})\)

by (auto elim:secComTyping.cases)

22
show ?case
proof (rule nonInterferenceI)
  fix s1 s2 s1' s2'
  assume Γ ⊢ s1 ≈_L s2 and ⟨V:=e,s1⟩ ® ⟨Skip,s1⟩ and ⟨V:=e,s2⟩ ®
  ⟨Skip,s2⟩
  from ⟨V:=e,s1⟩ ® ⟨Skip,s1⟩ have s1' = s1(V:=e] s1) by (auto intro:LAss-reds)
  from ⟨V:=e,s2⟩ ® ⟨Skip,s2⟩ have s2' = s2(V:=e] s2) by (auto intro:LAss-reds)
  from varprem show Γ ⊢ s1' ≈_L s2'
  proof
    assume Γ V = Some High
    with Γ ⊢ s1 ≈_L s2 (s1' = s1(V:=e] s1)) (s2' = s2(V:=e] s2))
    show ?thesis by (auto intro:assignNHighlemma)
  next
    assume Γ V = Some Low ∧ Γ ⊢ e : Low ∧ T = Low
    with Γ ⊢ s1 ≈_L s2 (s1' = s1(V:=e] s1)) (s2' = s2(V:=e] s2))
    show ?thesis by (auto intro:assignNLowlemma)
  qed
qed
next
  case (Seq c1 c2)
  note IH1 = ⟨∧T. Γ,T ⊢ c1 ⟹ nonInterference Γ c1⟩
  note IH2 = ⟨∧T. Γ,T ⊢ c2 ⟹ nonInterference Γ c2⟩
  show ?case
  proof (cases T)
    case High
    with Γ,T ⊢ c1;;c2 have Γ,High ⊢ c1 and Γ,High ⊢ c2
    by (auto elim:secComTyping.cases)
    from IH1[OF ⟨Γ,High ⊢ c1⟩] have nonInterference Γ c1.
    moreover
    from IH2[OF ⟨Γ,High ⊢ c2⟩] have nonInterference Γ c2.
    ultimately show ?thesis by (auto intro:SeqCompositionality)
  next
    case Low
    with Γ,T ⊢ c1;;c2 have (Γ,Low ⊢ c1 ∧ Γ,Low ⊢ c2) ∨ Γ,High ⊢ c1;;c2
    by (auto elim:secComTyping.cases)
    thus ?thesis
  proof
    assume Γ,Low ⊢ c1 ∧ Γ,Low ⊢ c2
    hence Γ,Low ⊢ c1 and Γ,Low ⊢ c2 by simp-all
    from IH1[OF ⟨Γ,Low ⊢ c1⟩] have nonInterference Γ c1.
    moreover
    from IH2[OF ⟨Γ,Low ⊢ c2⟩] have nonInterference Γ c2.
    ultimately show ?thesis by (auto intro:SeqCompositionality)
  next
    assume Γ,High ⊢ c1;;c2
    hence Γ,High ⊢ c1 and Γ,High ⊢ c2 by (auto elim:secComTyping.cases)
from IH1[OF Γ,High ⊢ c1] have nonInterference Γ c1.
moreover
from IH2[OF Γ,High ⊢ c2] have nonInterference Γ c2.
ultimately show thesis by(auto intro:SeqCompositionality)
qed
next

case Cond b c1 c2
note IH1 = ⟨⋀T. Γ,T ⊢ c1 ⇒ nonInterference Γ c1⟩
note IH2 = ⟨⋀T. Γ,T ⊢ c2 ⇒ nonInterference Γ c2⟩
show thesis
proof (cases T)
  case High
  with ⟨Γ,T ⊢ if (b)c1 else c2⟩ show thesis
  by(auto intro:CondHighCompositionality)
next
  case Low
  with ⟨Γ,T ⊢ if (b)c1 else c2⟩
  have (Γ ⊢ b : Low ∧ Γ,Low ⊢ c1 ∧ Γ,Low ⊢ c2) ∨ Γ,High ⊢ if (b)c1 else c2
  by(auto elim:secComTyping_cases)
  thus thesis
  proof
    assume Γ,High ⊢ if (b)c1 else c2
    thus thesis
    by(auto intro:CondHighCompositionality)
  qed
next

case While b c'
note IH = ⟨⋀T. Γ,T ⊢ c' ⇒ nonInterference Γ c'⟩
show thesis
proof (cases T)
  case High
  with ⟨Γ,T ⊢ while (b)c'} show thesis
  by(auto intro:WhileHighCompositionality)
next
  case Low
  with ⟨Γ,T ⊢ while (b)c'}
  have (Γ ⊢ b : Low ∧ Γ,Low ⊢ c'} ∨ Γ,High ⊢ while (b)c'
  by(auto elim:secComTyping_cases)
  thus thesis
  proof
    assume Γ ⊢ b : Low ∧ Γ,Low ⊢ c'}

hence $\Gamma \vdash b : \text{Low}$ and $\Gamma, \text{Low} \vdash c'$ by simp-all

moreover

from IH[OF $\Gamma, \text{Low} \vdash c'$] have nonInterference $\Gamma \vdash c'$.

ultimately show thesis by(auto intro:WhileLowCompositionality)

next

assume $\Gamma, \text{High} \vdash \text{while (b) c'}$

thus thesis by(auto intro:WhileHighCompositionality)

qed

end

theory Execute
imports secTypes
begin

3 Executing the small step semantics

code-pred (modes: $i \Rightarrow o \Rightarrow \text{bool}$ as exec-red, $i \Rightarrow i \Rightarrow o \Rightarrow \text{bool}$, $i \Rightarrow o \Rightarrow \text{bool}$, $i \Rightarrow i \Rightarrow \text{bool}$) red.

thm red.
equation
definition [code]: one-step $x = \text{Predicate.the (exec-red } x)$

lemmas [code-pred-intro] = typeVal[where $\text{lev = Low}$] typeVal[where $\text{lev = High}$] typeVar

<table>
<thead>
<tr>
<th>$\text{typeBinOp1}$</th>
<th>$\text{typeBinOp2}$</th>
<th>$\text{typeBinOp3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{where lev = Low}$</td>
<td>$\text{where lev = High}$</td>
<td>$\text{where lev = Low}$</td>
</tr>
</tbody>
</table>

code-pred (modes: $i \Rightarrow i \Rightarrow o \Rightarrow \text{bool}$ as compute-secExprTyping, $i \Rightarrow i \Rightarrow i \Rightarrow \text{bool}$ as check-secExprTyping) secExprTyping

proof –
case secExprTyping
from secExprTyping.prems show thesis

proof
fix $\Gamma \text{ V lev}$ assume $x = \Gamma \text{ xa = Val V xb = lev}$
from secExprTyping(1 - 2) this show thesis by (cases lev) auto

next
fix $\Gamma \text{ Vn lev}$
assume $x = \Gamma \text{ xa = Var Vn xb = lev \Gamma \text{ Vn = Some lev}$
from secExprTyping(3) this show thesis by (auto simp add: Predicate.eq-is-eq)

next
fix $\Gamma \text{ c1 c2 bop}$
assume $x = \Gamma \text{ xa = c1<bop> c2 xb = Low}$
$\Gamma \vdash c1 : \text{Low} \Gamma \vdash c2 : \text{Low}$
from secExprTyping(4) this show thesis by auto

next
fix $\Gamma$ $e_1$ $e_2$ $lev$ $bop$
assume $x = \Gamma$ $xa = e_1 < bop > e_2$ $xb = High$
$\Gamma \vdash e_1 : High$ $\Gamma \vdash e_2 : lev$
from $secExprTyping(5-6)$ this show thesis by (cases $lev$) (auto)
next
fix $\Gamma$ $e_1$ $e_2$ $lev$ $bop$
assume $x = \Gamma$ $xa = e_1 < bop > e_2$ $xb = High$
$\Gamma \vdash e_1 : lev$ $\Gamma \vdash e_2 : High$
from $secExprTyping(6-7)$ this show thesis by (cases $lev$) (auto)
qed

lemmas [code-pred-intro] = typeSkip[where $T$=Low] typeSkip[where $T$=High]
typeAssH[where $T$ = Low] typeAssH[where $T$ = High]
typeAssL typeSeq typeWhile typeIf typeConvert

code-pred (modes: $i => o => i =>$ bool as compute-secComTyping,
$i => i => i =>$ bool as check-secComTyping) secComTyping

proof –
case secComTyping
from secComTyping.prems show thesis
proof
fix $\Gamma$ $T$ assume $x = \Gamma$ $xa = T$ $xb = Skip$
from $secComTyping(1-2)$ this show thesis by (cases $T$) auto
next
fix $\Gamma$ $V$ $T$ $e$ assume $x = \Gamma$ $xa = T$ $xb = V := e$ $\Gamma$ $V$ = Some $High$
from $secComTyping(3-4)$ this show thesis by (cases $T$) (auto)
next
fix $\Gamma$ $e$ $V$
assume $x = \Gamma$ $xa = Low$ $xb = V := e$ $\Gamma$ $\vdash e : Low$ $\Gamma$ $V$ = Some $Low$
from $secComTyping(5)$ this show thesis by auto
next
fix $\Gamma$ $T$ $c_1$ $c_2$
assume $x = \Gamma$ $xa = T$ $xb = Seq$ $c_1$ $c_2$ $\Gamma, T \vdash c_1, T \vdash c_2$
from $secComTyping(6)$ this show thesis by auto
next
fix $\Gamma$ $b$ $T$ $c$
assume $x = \Gamma$ $xa = T$ $xb = while (b) c$ $\Gamma \vdash b : T, T \vdash c$
from $secComTyping(7)$ this show thesis by auto
next
fix $\Gamma$ $b$ $T$ $c_1$ $c_2$
assume $x = \Gamma$ $xa = T$ $xb = if (b) c_1$ else $c_2$ $\Gamma \vdash b : T, T \vdash c_1, T \vdash c_2$
from $secComTyping(8)$ this show thesis by blast
next
fix $\Gamma$ $c$
assume $x = \Gamma$ $xa = Low$ $xb = c$ $\Gamma, High \vdash c$
from $secComTyping(9)$ this show thesis by blast
qed

qed

26
3.1 An example taken from Volpano, Smith, Irvine

\[\text{definition } \text{com} = \text{if } (\text{Var } "x" \ll \text{Eq} \gg \text{Val} \ (\text{Intg} \ 1)) \text{ then } ("y" := \text{Val} \ (\text{Intg} \ 0)) \text{ else } ("y" := \text{Val} \ (\text{Intg} \ 0))\]

\[\text{definition } \text{Env} = \text{map-of } [("x", \text{High}), ("y", \text{High})]\]

\[\text{values } \{T. \text{Env} \vdash (\text{Var } "x" \ll \text{Eq} \gg \text{Val} \ (\text{Intg} \ 1)): T\}\]

\[\text{value } \text{Env}, \text{High} \vdash \text{com}\]

\[\text{value } \text{Env}, \text{Low} \vdash \text{com}\]

\[\text{values } 1 \{T. \text{Env}, T \vdash \text{com}\}\]

\[\text{definition } \text{Env}' = \text{map-of } [("x", \text{Low}), ("y", \text{High})]\]

\[\text{value } \text{Env}', \text{Low} \vdash \text{com}\]

\[\text{value } \text{Env}', \text{High} \vdash \text{com}\]

\[\text{values } 1 \{T. \text{Env}, T \vdash \text{com}\}\]

References
